

Référence rapide Go

Syntaxe, types, concurrence, gestion des erreurs

Bases

Hello World

```
package main
import "fmt"
func main() {
    fmt.Println("Hello, World!")
}
```

Exécuter et compiler

```
go run main.go      # compile and run
go build -o app .   # compile to binary
go test ./...       # run all tests
```

Initialisation de module

```
go mod init github.com/user/project
go mod tidy      # sync dependencies
```

Variables et types

Déclaration

```
var name string = "Go"
age := 15          // short declaration
var x, y int = 1, 2
const Pi = 3.14159
```

Types de base

bool	true, false
string	Séquence d'octets UTF-8 immuable
int, int8..int64	Entiers signés (plateforme / largeur fixe)
uint, uint8..uint64	Entiers non signés
float32, float64	Virgule flottante IEEE-754
byte	Alias pour uint8
rune	Alias pour int32 (point de code Unicode)

Valeurs zéro

int, float	0
bool	false
string	"" (chaîne vide)
pointer, slice, map	nil

Fonctions

Fonction de base

```
func add(a, b int) int {
    return a + b
}
```

Valeurs de retour multiples

```
func divide(a, b float64) (float64, error) {
    if b == 0 {
        return 0, errors.New("division by zero")
    }
    return a / b, nil
}
```

Variadique et anonyme

```
func sum(nums ...int) int {
    total := 0
    for _, n := range nums { total += n }
    return total
}
double := func(x int) int { return x * 2 }
```

Defer

```
func readFile(path string) {
    f, _ := os.Open(path)
    defer f.Close() // runs when function returns
}
```

Flux de contrôle

If / Else

```
if x > 0 {
    fmt.Println("positive")
} else if x == 0 {
    fmt.Println("zero")
} else {
    fmt.Println("negative")
}
```

Boucle for

```
for i := 0; i < 10; i++ { } // classic
for x < 100 { x *= 2 }     // while-style
for { break }              // infinite
for i, v := range slice { } // range
```

Switch

```
switch day {
case "Mon", "Tue":
    fmt.Println("early week")
case "Fri":
    fmt.Println("TGIF")
default:
    fmt.Println("other")
}
```

Structs et méthodes

Définition d'une struct

```
type User struct {
    Name string
    Email string
    Age int
}
u := User{Name: "Alice", Email: "a@b.com", Age: 30}
```

Méthodes

```
func (u User) Greeting() string {
    return "Hi, " + u.Name
}
func (u *User) SetAge(age int) {
    u.Age = age // pointer receiver mutates
}
```

Embedding

```
type Admin struct {
    User // embedded struct
    Level string
}
a := Admin{User: User{Name: "Bob"}, Level: "super"}
fmt.Println(a.Name) // promoted field
```

Interfaces

Définir et implémenter

```
type Stringer interface {
    String() string
}
// implicit implementation - no "implements" keyword
func (u User) String() string {
    return u.Name
}
```

Interfaces courantes

io.Reader	Read(p []byte) (n int, err error)
io.Writer	Write(p []byte) (n int, err error)
fmt.Stringer	String() string
error	Error() string

Assertion de type

```
var i interface{} = "hello"
s, ok := i.(string) // ok == true
switch v := i.(type) {
case string: fmt.Println(v)
case int:    fmt.Println(v * 2)
}
```

Goroutines et channels

Goroutines

```
go func() {
    fmt.Println("running concurrently")
}()
time.Sleep(time.Second)
```

Channels

```
ch := make(chan int) // unbuffered
buf := make(chan int, 5) // buffered
ch <- 42 // send
val := <-ch // receive
```

Select

```
select {
case msg := <-ch1:
    fmt.Println(msg)
case ch2 <- 42:
    fmt.Println("sent")
case <-time.After(time.Second):
    fmt.Println("timeout")
}
```

Patterns

sync.WaitGroup	Attendre que plusieurs goroutines se terminent
sync.Mutex	Verrou d'exclusion mutuelle pour l'état partagé
context.Context	Annulation, délais, valeurs liées à la requête

Gestion des erreurs

Pattern de base

```
result, err := doSomething()
if err != nil {
    return fmt.Errorf("failed: %w", err)
}
```

Erreurs personnalisées

```
type NotFoundError struct {
    ID string
}
func (e *NotFoundError) Error() string {
    return "not found: " + e.ID
}
```

Package errors

errors.New(msg)	Créer une erreur simple
fmt.Errorf("%w", err)	Encapsuler une erreur avec contexte
errors.Is(err, target)	Vérifier la chaîne d'erreurs pour une correspondance
errors.As(err, &target)	Extraire une erreur typée de la chaîne

Référence rapide Go

Slices et maps

Slices

```
s := []int{1, 2, 3}
s = append(s, 4, 5)
sub := s[1:3] // [2, 3]
cp := make([]int, len(s))
copy(cp, s)
```

Maps

```
m := map[string]int{"a": 1, "b": 2}
m["c"] = 3
val, ok := m["a"] // ok == true
delete(m, "b")
for k, v := range m { }
```

Opérations sur les slices

len(s)	Nombre d'éléments
cap(s)	Capacité du tableau sous-jacent
append(s, elems...)	Ajouter des éléments, peut réallouer
copy(dst, src)	Copier des éléments entre slices
slices.Sort(s)	Trier une slice (package slices Go 1.21+)

Paquets et imports

Styles d'import

```
import "fmt"
import (
    "os"
    "strings"
    "github.com/user/pkg"
)
```

Visibilité

Première lettre majuscule = exporté (public).
Première lettre minuscule = non exporté (privé au package).
Aucun mot-clé public/private nécessaire.

Bibliothèque standard courante

fmt	E/S formatées (Print, Sprintf, Errorf)
os	Fonctions OS (fichiers, env, args)
io	Primitives d'E/S (Reader, Writer)
net/http	Client et serveur HTTP
encoding/json	Encodage/décodage JSON
strings	Fonctions de manipulation de chaînes
strconv	Conversions chaîne ↔ nombre
testing	Framework de tests unitaires

Génériques

Paramètres de type

```
func Map[T, U any](s []T, f func(T) U) []U {
    r := make([]U, len(s))
    for i, v := range s { r[i] = f(v) }
    return r
}
```

Contraintes

```
type Number interface {
    ~int | ~float64
}
func Sum[T Number](nums []T) T {
    var total T
    for _, n := range nums { total += n }
    return total
}
```

Tests

Test de base

```
// file: math_test.go
func TestAdd(t *testing.T) {
    got := Add(2, 3)
    if got != 5 {
        t.Errorf("Add(2,3) = %d, want 5", got)
    }
}
```

Commandes de test

go test	Exécuter les tests du package courant
go test ./...	Exécuter tous les tests récursivement
go test -v	Sortie détaillée
go test -run TestAdd	Exécuter un test spécifique par nom
go test -bench .	Exécuter les benchmarks
go test -cover	Afficher le pourcentage de couverture